Технології Computer Vision
(Сертифікатна програма Data Science із Sigma Software)

**Міністерство освіти і науки України**
**Національний технічний університет України «КПІ» імені Ігоря Сікорського**
**Кафедра обчислювальної техніки ФІОТ**

ЗВІТ
з лабораторної роботи №1
з навчальної дисципліни «Технології Computer Vision»

Тема:

# ДОСЛІДЖЕННЯ ТЕХНОЛОГІЙ ПОБУДОВИ ТА ПЕРЕТВОРЕННЯ КООРДИНАТ ПЛОЩИННИХ (2D) ТА ПРОСТОРОВИХ (3D) ОБ’ЄКТІВ

Виконав ІП-31 Бойко Станіслав Сергійович

Перевірив Баран Данило Романович

Київ 2025

Технології Computer Vision
(Сертифікатна програма Data Science із Sigma Software)

**І. Мета:**
Виявити дослідити та узагальнити особливості формування та перетворення координат площинних (2d) та просторових (3d) об’єктів.

**ІІ. Завдання:**
Завдання І рівня складності – максимально 8 балів.
Здійснити синтез математичних моделей та розробити програмний скрипт, що реалізує базові операції 2D перетворень над геометричними примітивами. Для розробки використовувати матричні операції та технології композиційних перетворень. Вхідна матриця координат кутів геометричної фігури має бути розширеною. Функціонал скрипта, що розробляється має реалізувати технічних вимог табл.1 Додатку 1.

Завдання ІІ рівня – максимально 9 балів.
Здійснити синтез математичних моделей та розробити програмний скрипт, що реалізує базові операції 3D перетворень над геометричними примітивами: аксонометрична проекція будь-якого типу та з циклічне обертання (анімація) 3D графічного об’єкту навколо будь-якої обраної внутрішньої віссю. Траєкторію обертання не відображати. Для розробки використовувати матричні операції. Вхідна матриця координат кутів геометричної фігури має бути розширеною. Функціонал скрипта, що розробляється має реалізувати технічних вимог табл.2 Додатку 1.

Завдання ІІІ рівня – максимально 10 балів.
Реалізувати розробку програмного скрипта із функціоналом І та ІІ рівнів складності.

Обрано завдання III рівня.

| | | |
|---|---|---|
| 4 | Реалізувати операції: **обертання – переміщення – масштабування**. **3.** операцію реалізувати циклічно, траєкторію зміни положення цієї операції **відобразити.** Обрати самостійно: бібліотеку, розмір графічного вікна, розмір фігури, параметри реалізації операцій, кольорову гамму усіх графічних об'єктів. Всі операції перетворень мають здійснюватись у межах графічного вікна. | Ромб |

| | | |
|---|---|---|
| 4 | **Динаміка фігури**: графічна фігура з'являється та гасне, змінює колір заливки. **Обрати самостійно:** бібліотеку, розмір графічного вікна, розмір фігури, параметри зміни положення фігури, кольорову гамму усіх графічних об'єктів. Всі операції перетворень мають здійснюватись у межах графічного вікна. | Паралелепіпед |

## III. Результати виконання лабораторної роботи.

### 3.1. Синтезована математична модель;

Синтезована математична модель в даній лабораторній роботі для 2D графічних об'єктів включає в себе три основні компоненти перетворень:

1. Модель обертання (Rotation)

Матриця обертання на кут θ навколо довільної точки (cx, cy):

Композитне перетворення:

$R\_composite = T_1 \times R \times T_2$

Де:

$T_1$ - матриця переносу в початок координат:

$$T_1 = \begin{bmatrix} 1 & 0 & -cx \\ 0 & 1 & -cy \\ 0 & 0 & 1 \end{bmatrix}$$

R - матриця обертання:

$$R = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$T_2$ - матриця зворотного переносу:

$$T_2 = \begin{bmatrix} 1 & 0 & cx \\ 0 & 1 & cy \\ 0 & 0 & 1 \end{bmatrix}$$

Результуючі координати: $P' = P \times T_1 \times R \times T_2$

Математичне розкриття:

$$[x', y', 1] = [x, y, 1] \times T_1 \times R \times T_2$$

$$x' = cos(\theta) \times (x - cx) - sin(\theta) \times (y - cy) + cx$$
$$y' = sin(\theta) \times (x - cx) + cos(\theta) \times (y - cy) + cy$$

2. Модель переміщення (Translation)

Матриця переміщення на вектор (dx, dy):

$$T = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

Результуючі координати: $P' = P \times T$

Математичне розкриття:

$$[x', y', 1] = [x, y, 1] \times \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & \text{dy} \\ 0 & 0 & 1 \end{bmatrix}$$

$$x' = x \times 1 + y \times 0 + 1 \times dx = x + dx$$
$$y' = x \times 0 + y \times 1 + 1 \times dy = y + dy$$

3. Модель масштабування (Scaling)

Матриця масштабування з коефіцієнтами (sx, sy) відносно центра (cx, cy):

Композитне перетворення:

$S\_composite = T_1 \times S \times T_2$

Де:

S - матриця масштабування:

$$S = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Результуючі координати: $P' = P \times T_1 \times S \times T_2$

Математичне розкриття:

$$[x', y', 1] = [x, y, 1] \times T_1 \times S \times T_2$$

$$x' = sx \times (x - cx) + cx$$

$y' = sy \times (y - cy) + cy$

4. Модель динамічного масштабування

Коефіцієнт масштабування змінюється за синусоїдальним законом:

$$factor(t) = 1 + scale\_step \times sin(2\pi t/T)$$

Де:

– t - поточний крок анімації

– T - загальна кількість кроків

– *scale_step* - амплітуда зміни масштабу

5. Гомогенні координати

Всі точки представлені у гомогенних координатах:

$$P = [x, y, 1]$$

Це дозволяє представити всі афінні перетворення у вигляді матричних операцій та легко їх композувати.

6. Композиція перетворень

Загальна модель перетворення об'єкта на кожному кроці анімації:

$$P'(t) = P \times R(\theta(t)) \times T(dx, dy) \times S(factor(t))$$

Де:

– $\theta(t) = angle\_step \times t$ - кут обертання

– factor(t) - динамічний коефіцієнт масштабування

– (dx, dy) - вектор переміщення

7. Параметри моделі

– Кроки анімації: 60

– Крок обертання: 6° на ітерацію

– Вектор переміщення: (4, 2) пікселі

– Амплітуда масштабування: 0.05

– Початкові розміри ромба: 140×80 пікселів

Ця модель забезпечує плавну анімацію з одночасним обертанням, переміщенням та пульсацією розміру графічного об'єкта.

Тепер перейдемо до другого завдання. Синтезована математична модель в даній лабораторній роботі для 3D графічних об'єктів включає в себе п'ять основних компонентів перетворень:

1. Модель переміщення (Translation)
Матриця переміщення на вектор (dx, dy, dz):

$$T = \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Результуючі координати: P' = P × T
Математичне розкриття:

$$[x', y', z', 1] = [x, y, z, 1] \times \begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x' = x + dx$$
$$y' = y + dy$$
$$z' = z + dz$$

2. Модель обертання (Rotation)
Матриці обертання навколо координатних осей:

Обертання навколо осі Y:

$$Ry = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Обертання навколо осі X:

$$Rx = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\varphi) & -sin(\varphi) & 0 \\ 0 & sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Результуючі координати: P' = P × R

3. Модель аксонометричної проекції

Аксонометрична проекція складається з двох послідовних обертань:

Композитне перетворення:

$$A = Ry(\varphi) \times Rx(\theta)$$

Де:

φ = 35° - кут обертання навколо осі Y

θ = 25° - кут обертання навколо осі X

Результуючі координати: P' = P × Ry(35°) × Rx(25°)

Математичне розкриття:

$$x' = cos(35°) \times x + sin(35°) \times z$$

$$y' = sin(25°) \times sin(35°) \times x + cos(25°) \times y - sin(25°) \times cos(35°) \times z$$

$$z' = -cos(25°) \times sin(35°) \times x + sin(25°) \times y + cos(25°) \times cos(35°) \times z$$

4. Модель ортогональної проекції

Матриця проекції 3D об'єкта на 2D площину:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Результуючі координати: P' = P × P_matrix

Математичне розкриття:

$$[x', y', 0, 1] = [x, y, z, 1] \times P$$

$$x' = x$$

$y' = y$

$z' = 0$ (відкидається)

5. Модель яскравості (Brightness)

Функція видимості контролює з'явлення та зникнення об'єкта:

brightness(t) = |sin(2πt/T)|

Функція яскравості контролює інтенсивність кольорів:

brightness > 0.9 → білий колір (максимальна яскравість)
brightness < 0.1 → чорний колір (мінімальна яскравість)
0.1 ≤ brightness ≤ 0.9 → звичайні кольори з відтінками

Умова кольору:
if brightness > 0.9:
  колір = білий
elif brightness < 0.1:
  колір = чорний
else:
  колір = базовий_колір (світлий/темний варіант)

6. Гомогенні координати

Всі точки представлені у 4D гомогенних координатах:

$P = [x, y, z, 1]$

Це дозволяє представити всі афінні перетворення у вигляді матричних операцій та легко їх композувати.

7. Композиція перетворень

Загальна модель перетворення об'єкта на кожному кроці анімації:

$P'(t) = P \times A(\varphi, \theta) \times Ry(\alpha(t)) \times T(dx, dy, dz) \times P\_matrix$

Де:

– A(φ, θ) - аксонометрична проекція (φ=35°, θ=25°)
– α(t) = 360° × t/T - кут обертання навколо осі Y
– T(dx, dy, dz) - переміщення в центр екрана

– P_matrix - ортогональна проекція на 2D

8. Параметри моделі

– Кроки анімації: 60

– Розміри паралелепіпеда: 120×80×60

– Кути аксонометрії: φ=35°, θ=25°

– Швидкість обертання: 360°/60 = 6° на кадр

– Швидкість анімації: 0.07 секунди на кадр

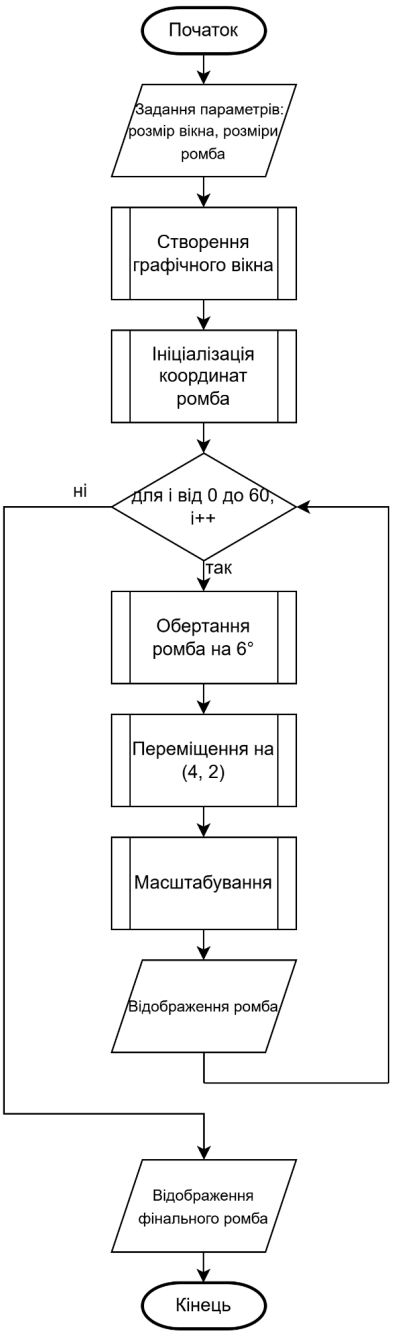
– Кількість граней: 6

– Кількість ребер: 12

– Кольорова гама: 8 базових кольорів + білий і чорний для екстремумів

"Ця модель забезпечує реалістичну 3D анімацію з аксонометричною проекцією, обертанням навколо вертикальної осі, циклічною зміною кольорів та ефектом пульсації яскравості від чорного до білого кольору з відображенням як заповнених граней, так і каркасної структури."

**3.2. Результати архітектурного проектування та їх опис;**




Опис алгоритму для першого завдання:

1. Початок програми.

2. Задання параметрів: встановлюється розмір графічного вікна 600×600 пікселів; задаються розміри ромба через його діагоналі (вертикальна diag_v = 140, горизонтальна diag_h = 80); визначається центр сцени як середина вікна; готуються параметри анімації: кількість кроків steps = 60, крок обертання angle_step = 6°, вектор переміщення (dx, dy) = (4, 2), амплітуда масштабування scale_step = 0.05.

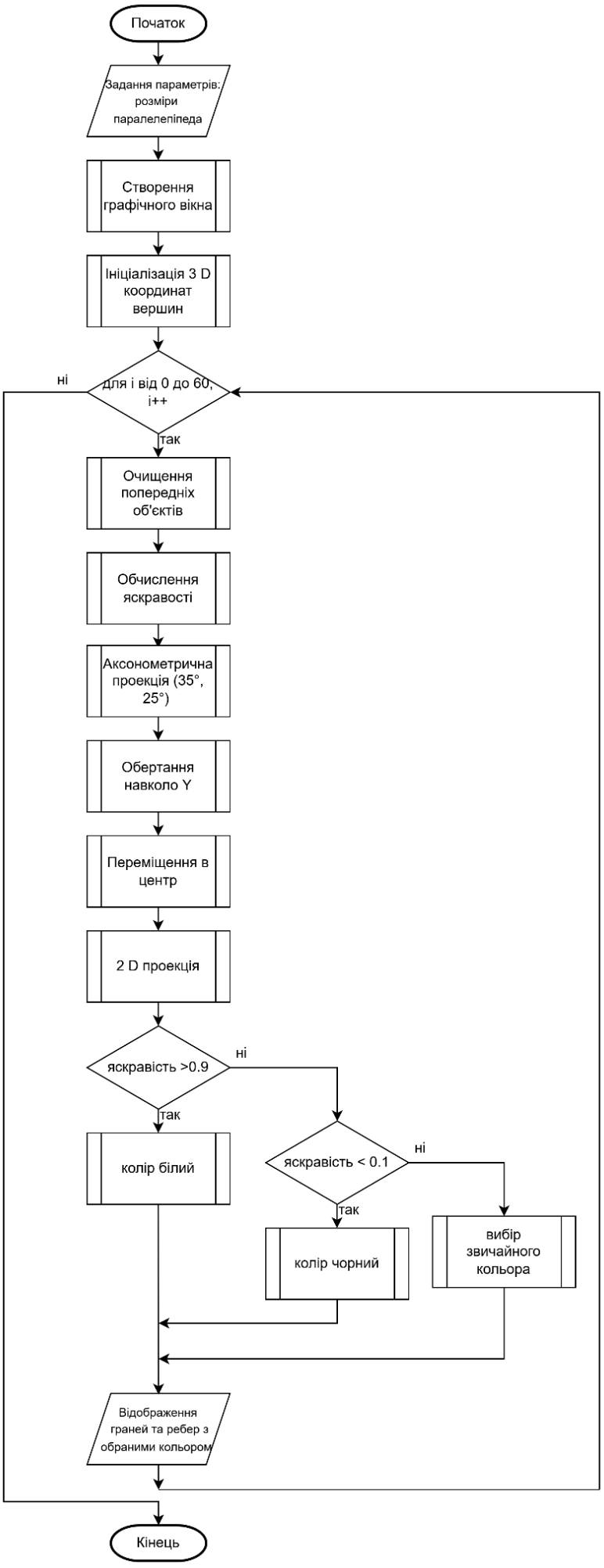
3. Створення графічного вікна: ініціалізується об’єкт вікна з білим фоном для відображення анімації.

4. Ініціалізація координат ромба: формується матриця вершин ромба у гомогенних координатах (4 точки), орієнтована по діагоналях і розміщена в центрі вікна; створюється робоча копія фігури для подальших перетворень.

5. Запуск циклу анімації на 60 ітерацій: на кожній ітерації обчислюється динамічний коефіцієнт масштабування factor = 1 + scale_step × sin(2π i / steps); до поточних координат ромба послідовно застосовуються три перетворення: 5.1. Обертання на 6° навколо поточного центру сцени за допомогою композиції матриць $T_1 \times R \times T_2$. 5.2. Переміщення на вектор (4, 2) з паралельним оновленням центру сцени (центр зсувається разом з ромбом). 5.3. Масштабування відносно поточного центру з коефіцієнтом factor по обох осях (S-композиція $T_1 \times S \times T_2$), що створює плавну пульсацію розміру. Після застосування перетворень ромб відображається: будуються точки, створюється полігон, задаються колір обводки (світло-блакитний) та заливки (жовтий), фігура малюється у вікні; виконується коротка пауза 0.07 с для плавності анімації.

6. Відображення фінального ромба: після завершення циклу останній стан фігури виділяється—ромб повторно малюється з червоною обводкою та помаранчевою заливкою, щоб зафіксувати фінальне положення.

7. Завершення програми: очікується клік мишею у вікні для завершення перегляду; вікно закривається; кінець виконання.


Початок
Задання параметрів: розміри паралелепіпеда
Створення графічного вікна
Ініціалізація 3 D координат вершин
для і від 0 до 60, і++
ні
так
Очищення попередніх об'єктів
Обчислення яскравості
Аксонометрична проекція (35°, 25°)
Обертання навколо Y
Переміщення в центр
2 D проекція
яскравість >0.9
ні
так
колір білий
яскравість < 0.1
ні
так
колір чорний
вибір звичайного кольора
Відображення граней та ребер з обраними кольором
Кінець

Опис алгоритму для другого завдання:

1. Початок програми.
2. Задання параметрів: задається розмір графічного вікна 600×600 пікселів, розміри паралелепіпеда по осях (size_x = 120, size_y = 80, size_z = 60), масив кольорів для заливки граней, кількість кроків анімації steps = 60. Визначається центр сцени як середина вікна, а також структури граней і ребер фігури у вигляді індексів вершин.
3. Створення графічного вікна: ініціалізується вікно відображення з білим фоном, у якому будуть малюватися всі кадри анімації.
4. Ініціалізація 3D координат вершин: формується матриця з 8 вершин паралелепіпеда у гомогенних координатах (формат [x, y, z, 1]), розташованих відносно центрованої системи координат об’єкта.
5. Ініціалізація лічильника і запуск циклу анімації на 60 кроків: перед початком кожного кроку з попереднього кадру видаляються намальовані об’єкти (грані та ребра), щоб забезпечити чисте відтворення нового кадру.
6. Обчислення яскравості: для поточного кроку розраховується параметр brightness = $|\sin(2\pi \cdot i/steps)|$, який плавно змінюється від 0 до 1 і керує вибором кольору граней і ребер (найтемніше — чорний, найяскравіше — білий, між ними — відповідні світлі/темні відтінки базового кольору).
7. Аксонометрична «орієнтація» об’єкта: до копії масиву вершин послідовно застосовуються обертання навколо осі Y на 35° і навколо осі X на 25°, що задає фіксовану аксонометричну орієнтацію для всієї анімації.
8. Обертання навколо внутрішньої осі: об’єкт додатково обертається навколо осі Y на кут angle = 360°·(i/steps), що забезпечує рівномірне повне обертання за один цикл анімації.
9. Переміщення в центр сцени: після обертань координати вершин зсуваються так, щоб об’єкт був розміщений у геометричному центрі вікна відображення.
10. 2D проекція: 3D координати проєктуються на 2D площину екрана ортогональною проекцією (Z-складова відкидається), утворюючи набір 2D точок для промальовування.
11. Вибір кольору за яскравістю: для кожної грані визначається колір заливки відповідно до значення brightness — при brightness > 0.9 встановлюється біла заливка, при brightness < 0.1 — чорна; у проміжних

значеннях обирається світлий або темний відтінок поточного базового кольору. Аналогічно для ребер обирається колір контурів (білий, чорний або сині відтінки).

12. Відображення граней та ребер: для кожної грані будується багатокутник із відповідних вершин, задається обраний колір заливки та чорний контур, після чого грань малюється у вікні. Далі послідовно промальовуються всі ребра лініями з кольором, обраним відповідно до яскравості.

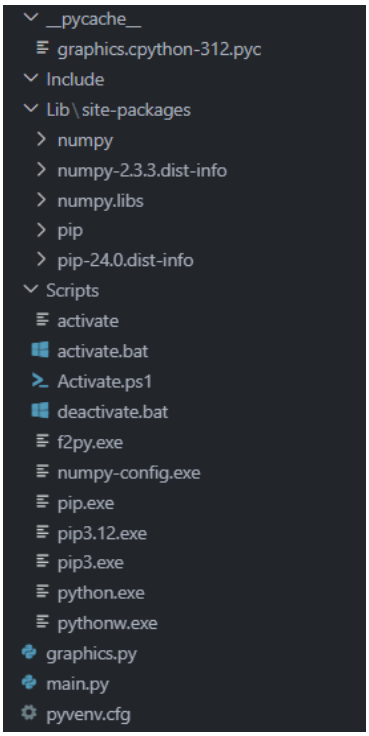
13. Пауза і інкремент кроку: виконується коротка пауза (близько 0.07 с) для плавності анімації, після чого цикл переходить до наступної ітерації, поки не буде виконано всі 60 кроків.

14. Завершення програми: після завершення анімації очікується клік мишею, щоб користувач міг закрити вікно; програма завершує роботу.

**3.3. Опис структури проекту програми;**

Для реалізації розробленого алгоритму мовою програмування Python з використанням можливостей інтегрованого середовища Visual Studio Code сформовано проект.

Проект базується на об'єктно-орієнтованій парадигмі та функціональному програмуванні і має таку структуру:




Опис файлів:

– main.py – основний файл програмного коду лабораторної роботи, що містить реалізацію двох завдань:

task1()

– функція для анімації 2D ромба з перетвореннями обертання, переміщення та масштабування

task2()

– функція для анімації 3D паралелепіпеда з аксонометричною проекцією

–graphics.py – модуль графічної бібліотеки для створення вікон та відображення геометричних об'єктів

– pyvenv.cfg – конфігураційний файл віртуального середовища Python

– Scripts/ – директорія з виконуваними файлами для активації віртуального середовища

– Lib/site-packages/ – директорія з встановленими бібліотеками:

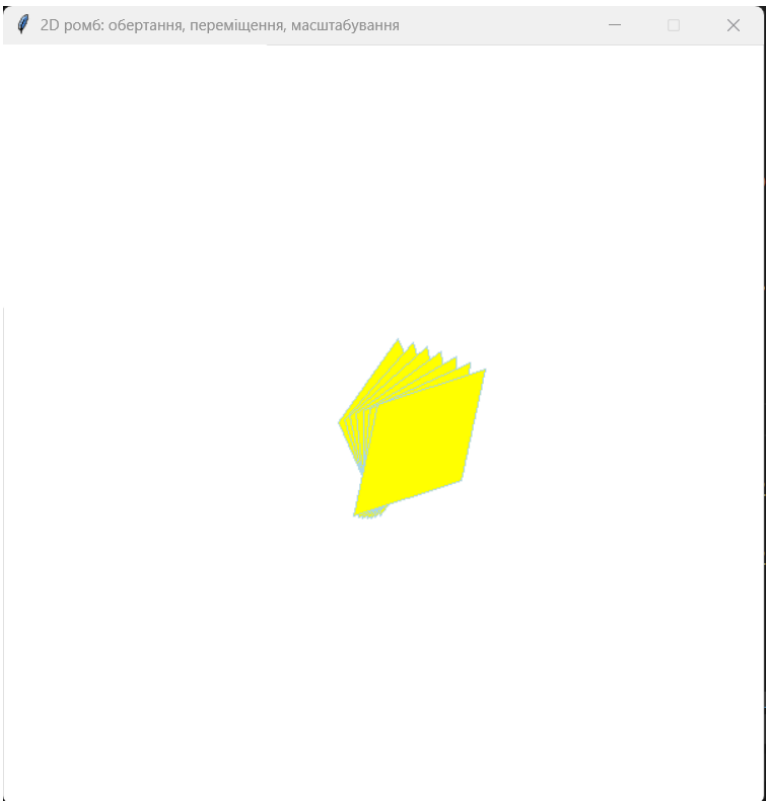
– numpy/ – бібліотека для роботи з масивами та математичними операціями

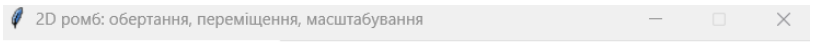
– pycache/ – директорія з компільованими Python-файлами для оптимізації швидкості виконання

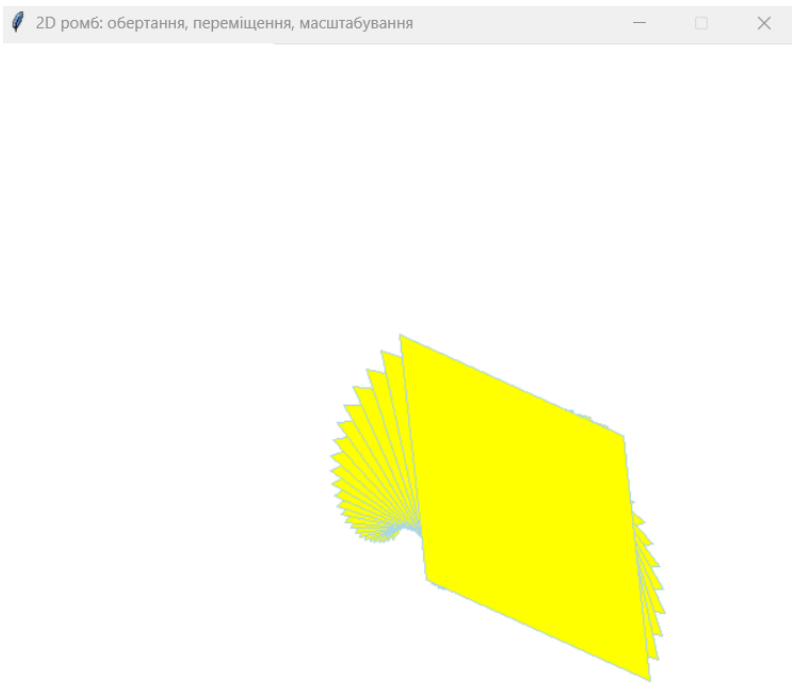
Проект використовує віртуальне середовище Python для ізоляції залежностей та забезпечення стабільної роботи програми.

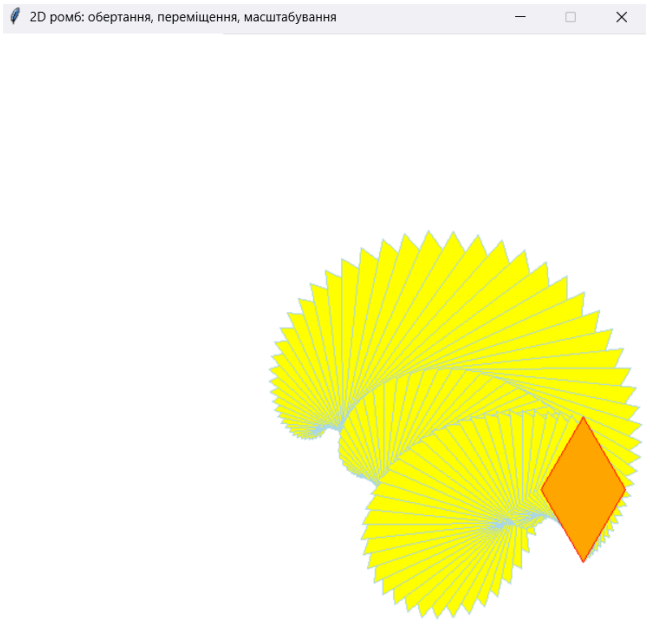
**3.4. Результати роботи програми відповідно до завдання (допускається у формі скриншотів);**

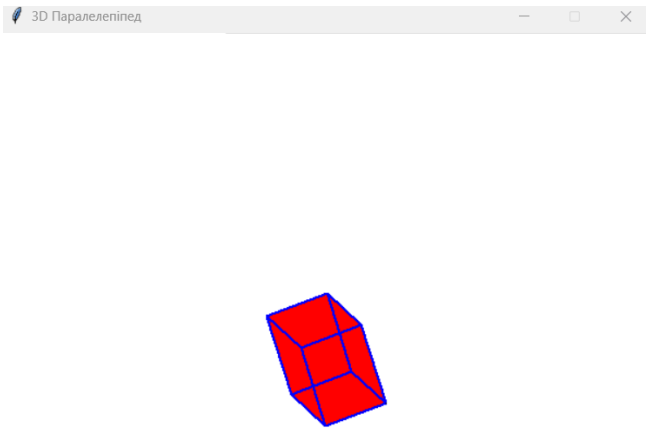
Далі зображені скріншоти для обох завдань.

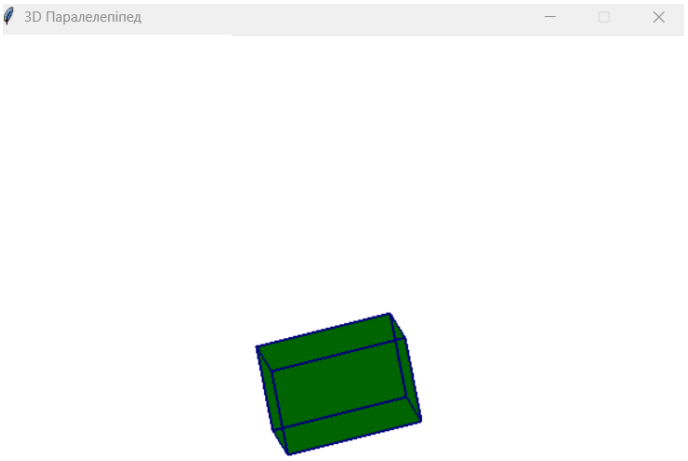

2D ромб: обертання, переміщення, масштабування

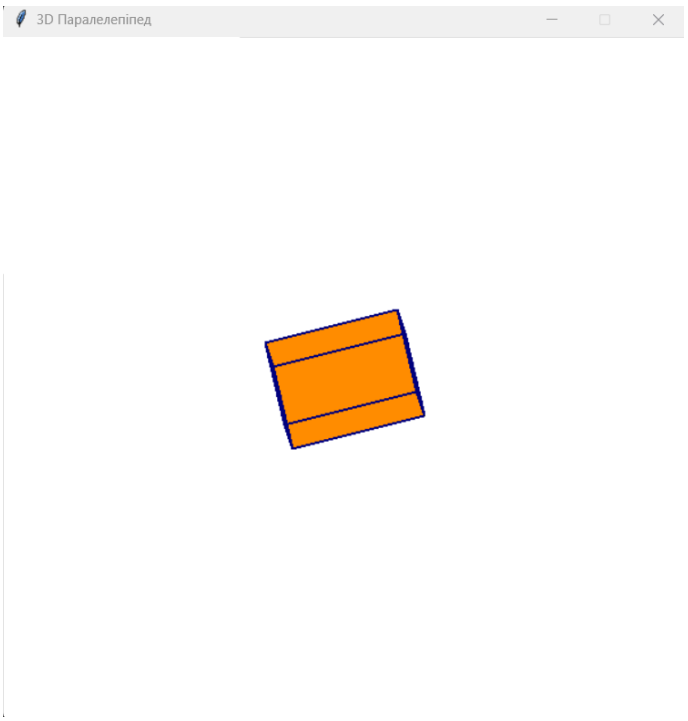

2D ромб: обертання, переміщення, масштабування


2D ромб: обертання, переміщення, масштабування


2D ромб: обертання, переміщення, масштабування


3D Паралелепіпед


3D Паралелепіпед




## 3.5. Програмний код, що забезпечує отримання результату (допускається у формі скриншотів).

https://github.com/StanislavBoiko31/Computer-Vision/tree/main/lab%201

```
from graphics import *
import numpy as np
import time
import math as mt

def task1():
    win_size = 600
    win = GraphWin("2D ромб: обертання, переміщення, масштабування", win_size, win_size)
    win.setBackground('white')

    center = np.array([win_size // 2, win_size // 2])
    diag_v = 140
    diag_h = 80


    diamond = np.array([
        [center[0], center[1] - diag_v // 2, 1],
        [center[0] + diag_h // 2, center[1], 1],
        [center[0], center[1] + diag_v // 2, 1],
        [center[0] - diag_h // 2, center[1], 1]
    ])

    def rotate(points, angle_deg, center):
        angle_rad = np.deg2rad(angle_deg)
        cx, cy = center
        T1 = np.array([[1, 0, -cx],
```

```
                [0, 1, -cy],
                [0, 0, 1]])
    R = np.array([[mt.cos(angle_rad), -mt.sin(angle_rad), 0],
                [mt.sin(angle_rad),  mt.cos(angle_rad), 0],
                [0, 0, 1]])
    T2 = np.array([[1, 0, cx],
                [0, 1, cy],
                [0, 0, 1]])
    return points @ T1.T @ R.T @ T2.T

def translate(points, dx, dy):
    T = np.array([[1, 0, dx],
                [0, 1, dy],
                [0, 0, 1]])
    return points @ T.T



def scale(points, sx, sy, center):
    cx, cy = center
    T1 = np.array([[1, 0, -cx],
                [0, 1, -cy],
                [0, 0, 1]])
    S = np.array([[sx, 0, 0],
                [0, sy, 0],
                [0, 0, 1]])
    T2 = np.array([[1, 0, cx],
                [0, 1, cy],
                [0, 0, 1]])
    return points @ T1.T @ S.T @ T2.T

steps = 60
dx, dy = 4, 2
angle_step = 6
scale_step = 0.05
current_diamond = diamond.copy()
trajectory_color = 'lightblue'

for i in range(steps):
    factor = 1 + scale_step * mt.sin(i * 2 * mt.pi / steps)
    current_diamond = rotate(current_diamond, angle_step, center)
    current_diamond = translate(current_diamond, dx, dy)
    center += np.array([dx, dy])
    current_diamond = scale(current_diamond, factor, factor, center)

    pts = [Point(*current_diamond[j, :2]) for j in range(4)]
    poly = Polygon(*pts)
    poly.setOutline(trajectory_color)
    poly.setFill('yellow')
    poly.draw(win)
    time.sleep(0.07)
```

```
    pts = [Point(*current_diamond[j, :2]) for j in range(4)]
    poly = Polygon(*pts)
    poly.setOutline('red')
    poly.setFill('orange')
    poly.draw(win)

    win.getMouse()
    win.close()

def task2():
    win_size = 600
    win = GraphWin("3D Паралелепіпед", win_size, win_size)
    win.setBackground('white')

    size_x, size_y, size_z = 120, 80, 60
    center = np.array([win_size // 2, win_size // 2, win_size // 2])
    cube = np.array([
        [-size_x/2, -size_y/2, -size_z/2, 1],
        [ size_x/2, -size_y/2, -size_z/2, 1],
        [ size_x/2,  size_y/2, -size_z/2, 1],
        [-size_x/2,  size_y/2, -size_z/2, 1],
        [-size_x/2, -size_y/2,  size_z/2, 1],
        [ size_x/2, -size_y/2,  size_z/2, 1],
        [ size_x/2,  size_y/2,  size_z/2, 1],
        [-size_x/2,  size_y/2,  size_z/2, 1],
    ])

    def translate(points, dx, dy, dz):
        T = np.array([[1, 0, 0, dx],
                      [0, 1, 0, dy],
                      [0, 0, 1, dz],
                      [0, 0, 0, 1]])
        return points @ T.T

    def rotate(points, angle_deg, axis='y'):
        angle_rad = np.deg2rad(angle_deg)
        if axis == 'x':
            R = np.array([[1, 0, 0, 0],
                          [0, mt.cos(angle_rad), -mt.sin(angle_rad), 0],
                          [0, mt.sin(angle_rad),  mt.cos(angle_rad), 0],
                          [0, 0, 0, 1]])
        elif axis == 'y':
            R = np.array([[mt.cos(angle_rad), 0, mt.sin(angle_rad), 0],
                          [0, 1, 0, 0],
                          [-mt.sin(angle_rad), 0, mt.cos(angle_rad), 0],
                          [0, 0, 0, 1]])
        elif axis == 'z':
            R = np.array([[mt.cos(angle_rad), -mt.sin(angle_rad), 0, 0],
                          [mt.sin(angle_rad),  mt.cos(angle_rad), 0, 0],
                          [0, 0, 1, 0],
                          [0, 0, 0, 1]])
        else:
```

```
        R = np.eye(4)
    return points @ R.T

def axonometric(points, phi_deg=45, theta_deg=30):
    points = rotate(points, phi_deg, axis='y')
    points = rotate(points, theta_deg, axis='x')
    return points

def project(points):
    P = np.array([[1, 0, 0, 0],
                  [0, 1, 0, 0],
                  [0, 0, 0, 0],
                  [0, 0, 0, 1]])
    return points @ P.T

def to_screen(points):
    return [Point(points[i,0], points[i,1]) for i in range(points.shape[0])]

faces = [
    [0,1,2,3],
    [4,5,6,7],
    [0,1,5,4],
    [2,3,7,6],
    [1,2,6,5],
    [0,3,7,4],
]

edges = [
    [0,1], [1,2], [2,3], [3,0],
    [4,5], [5,6], [6,7], [7,4],
    [0,4], [1,5], [2,6], [3,7]
]

steps = 60
colors = ['yellow', 'orange', 'lightgreen', 'lightblue', 'violet', 'pink', 'cyan', 'red']

drawn_objects = []

for i in range(steps):
    for obj in drawn_objects:
        obj.undraw()
    drawn_objects.clear()

    alpha = i / steps
    fill_color = colors[i % len(colors)]
    angle = 360 * alpha

    brightness = abs(mt.sin(alpha * 2 * mt.pi))

    obj = cube.copy()
    obj = axonometric(obj, phi_deg=35, theta_deg=25)
    obj = rotate(obj, angle, axis='y')
```

```
    obj = translate(obj, win_size//2, win_size//2, 0)
    obj2d = project(obj)

    for face in faces:
        pts = to_screen(obj2d[face])
        poly = Polygon(*pts)

        base_color = colors[i % len(colors)]
        if brightness > 0.9:
            bright_color = 'white'
        elif brightness < 0.1:
            bright_color = 'black'
        elif base_color == 'yellow':
            bright_color = 'yellow' if brightness > 0.5 else 'gold'
        elif base_color == 'orange':
            bright_color = 'orange' if brightness > 0.5 else 'darkorange'
        elif base_color == 'lightgreen':
            bright_color = 'lightgreen' if brightness > 0.5 else 'darkgreen'
        elif base_color == 'lightblue':
            bright_color = 'lightblue' if brightness > 0.5 else 'darkblue'
        elif base_color == 'violet':
            bright_color = 'violet' if brightness > 0.5 else 'darkviolet'
        elif base_color == 'pink':
            bright_color = 'pink' if brightness > 0.5 else 'deeppink'
        elif base_color == 'cyan':
            bright_color = 'cyan' if brightness > 0.5 else 'darkcyan'
        else:
            bright_color = 'red' if brightness > 0.5 else 'darkred'

        poly.setFill(bright_color)
        poly.setOutline('black')
        poly.draw(win)
        drawn_objects.append(poly)

    for edge in edges:
        pts = to_screen(obj2d[edge])
        line = Line(pts[0], pts[1])
        if brightness > 0.9:
            edge_color = 'white'
        elif brightness < 0.1:
            edge_color = 'black'
        else:
            edge_color = 'blue' if brightness > 0.5 else 'navy'
        line.setOutline(edge_color)
        line.setWidth(2)
        line.draw(win)
        drawn_objects.append(line)

    time.sleep(0.07)

win.getMouse()
win.close()
```

```
if __name__ == "__main__":
    task1()
    task2()
```

## IV. Висновки.

У ході виконання лабораторної роботи було успішно реалізовано завдання III рівня складності, що включало розробку програмного скрипта з функціоналом як 2D, так і 3D перетворень графічних об'єктів. Для 2D завдання було створено математичну модель ромба з використанням гомогенних координат та реалізовано композиційні перетворення обертання, переміщення та масштабування з анімацією у 60 кроків. Для 3D завдання було синтезовано математичну модель паралелепіпеда з аксонометричною проекцією, циклічним обертанням навколо вертикальної осі та ефектом пульсації яскравості від чорного до білого кольору.

Технічна реалізація базувалася на використанні мови програмування Python з бібліотекою NumPy для матричних операцій та Graphics.py для візуалізації.

Виконана робота забезпечила глибоке засвоєння теоретичних принципів роботи з гомогенними координатами, методів композиції матричних перетворень, особливостей аксонометричної та ортогональної проекції 3D об'єктів на 2D площину. Практичні навички включали реалізацію матричних операцій для графічних перетворень, створення плавних анімацій, роботу з багатовимірними масивами та управління графічними об'єктами.

Виконав: ІП-31 Бойко Станіслав